OPENALT 2026

ktapi.sys · LPE

Od čtení/zápisu fyzické paměti k SYSTEM

Vojtěch Hron

hron.re · @n00bDebugger

01 · O MNĚ

whoami

Vojtěch Hron

n00bDebugger

Student kyberbezpečnosti. Hlavně mě baví ofenzivní stránka - red team a vývoj exploitů.

Chci rozumět tomu, jak se věci rozbijí. Nic mi spolehlivěji neukáže, kde byly špatně postavené.

- Vulnerability hunting
- Exploit development
- Red team techniques
- Malware analysis

02 · PROČ TAHLE PŘEDNÁŠKA

Jak to celé začalo

Chtěl jsem napsat článek o **Living off the Land** na Windows - a potřeboval jsem **BYOVD** příklad.

První driver, který jsem si vybral, už byl v Microsoft blocklistu. Musel jsem najít jiný.

Vybral jsem **ktapi.sys** - podepsaný, málo známý, na LOLDrivers. A ponořil jsem se do něj.

Není to akademický problém: podobné zranitelné podepsané drivery dnes reálně používají **ransomware skupiny** k vypnutí EDR a eskalaci práv.

BYOVD = Bring Your Own Vulnerable Driver

03 · AGENDA

Program · 105 minut

#	Sekce	Čas	Co to je
1	Windows kernel - ringy, paměť, CR3, mitigace	~20 min	základy
2	Windows drivery - IOCTL, podpis, Ring 0	~15 min	základy
3	Útoky na drivery - BYOVD, LOLDrivers, ransomware	~15 min	kontext
4	Případ ktapi.sys - RE, page tables, token, PoC + demo	~40 min	hlavní
5	Obrana - HVCI, WDAC, LOLDrivers blocklist	~10 min	obrana
6	Dotazy	~5 min	Q&A

Dotazy prosím průběžně - čím dřív, tím lépe.

ČÁST 2 / 8

Jádro Windows

Krátký základ, abychom věděli, co **chceme napadnout** a co nás v tom bude **chránit**.

25 min · slajdy 5-16

KERNEL · RINGY

Ochranné kruhy CPU



Ring 0 - jádro
Plný přístup k HW, správa paměti, drivery. Žádná záchranná síť.

Ring 3 - user-space
Normální procesy. Jádro je přes syscally jediná brána dál.

Windows používá jen Ring 0 a Ring 3. Ring 1 a 2 jsou historický relikt.

Náš cíl: z Ring 3 manipulovat data jádra v Ring 0.

KERNEL · PAMĚŤ

Virtuální vs. fyzická paměť

// virtuální

Co vidí proces

- Každý proces má vlastní 48bitový adresní prostor
- Spodní polovina = user-space
- Horní polovina = kernel (sdílený)
- Adresa je jen „jméno“ do tabulky

// fyzická

Co vidí RAM

- Skutečné bajty v paměti
- Jedno RAM se dělí mezi všechny
- Pro přístup potřebuješ Ring 0
- Fyzická adresa klíče: **CR3**

Pointa: Kdo dokáže libovolně číst a zapisovat **fyzickou** paměť, drží v ruce všechno - jádro, všechny procesy, všechny stránky.

KERNEL · STRÁNKOVÁNÍ

K čemu stránkování

01

Izolace

Každý proces má svoje tabulky - vzájemně na sebe nevidí.

02

Ochrana

Flagy U/S, R/W, NX - stránka ví, co se s ní smí dělat.

03

Flexibilita

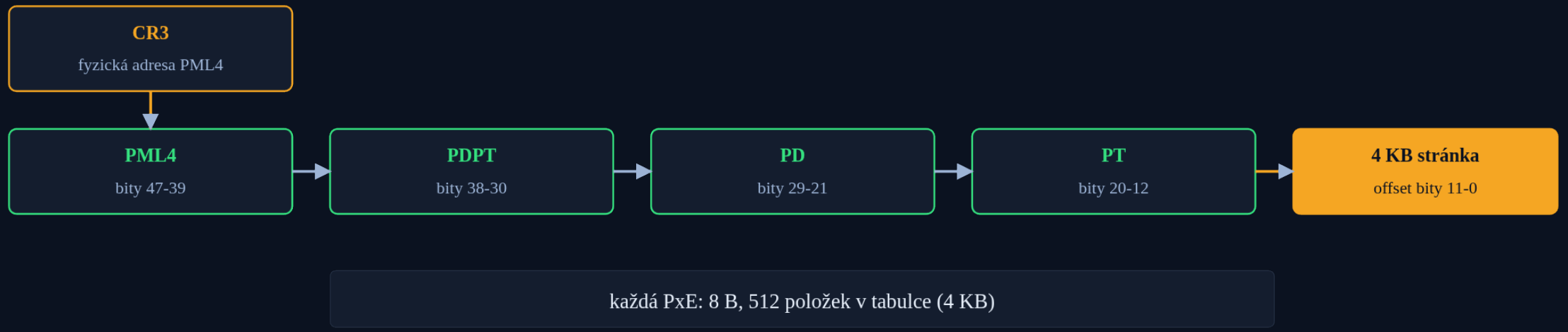
Swap, memory-mapping, COW - vše stojí na stránkách.

Jak: adresní prostor se dělí na stránky po **4 KB** (někdy 2 MB nebo 1 GB). CPU přes **CR3** + 4 úrovně tabulek převede virtuální adresu na fyzickou.

Stránkovací tabulky žijí ve fyzické paměti.

KERNEL · 4-LEVEL PAGING

x64 překlad adres: 4 úrovně



`4 úrovně × 9 bitů = 36 bitů indexů` `+ 12 bitů offset = 48 bitů`

Při zápisu PoC si tyhle tabulky projdeme ručně.

KERNEL · PTE

Co nese jeden záznam tabulky

Bit	Název	Co znamená
0	P (Present)	Stránka je skutečně v RAM.
1	R/W	0 = jen čtení, 1 = zápis povolen.
2	U/S	0 = jen Ring 0, 1 = i Ring 3 vidí.
7	PS (Page Size)	V PDPT/PD značí „velká“ stránka (1 GB / 2 MB).
12-51	PFN	Fyzická adresa dalšího levelu nebo stránky.
63	NX	1 = kód z této stránky se nespouští (DEP).

Když přepíšu NX nebo R/W ve fyz. paměti - stránka se chová jinak.

KERNEL · MITIGACE

Co stojí útočníkovi v cestě

DEP / NX
Nespustitelná data

KASLR
Náhodný layout jádra

SMEP
Kernel nespustí user kód

SMAP
Kernel nečte user data

DSE
Jen podepsané drivery

PatchGuard
Hlídá kritické struktury

HVCI
Hypervizor hlídá CI

VBS
Virtualization-based security

BYOVD →
obchází některé z nich

MITIGACE · DEP / NX

Data Execution Prevention

Co dělá

Hardwarový bit **NX** (bit 63 PTE) označí stránku jako nespustitelnou. CPU na ni odmítne skočit.

Proti čemu

Klasický shellcode: nacpat kód do heapu nebo zásobníku a skočit tam. S DEP = okamžitý crash.

Kde končí: když umím psát do PTE ve fyz. paměti, mohu bit NX smáznout. Nebo prostě využiju už existující spustitelný kód (ROP, ret2kernel).

V našem útoku NX bit vůbec neřešíme - kód neběží.

MITIGACE · KASLR

Kernel ASLR

Co dělá

Pozice **ntoskrnl.exe** a dalších modulů se při každém bootu posune o náhodný offset.

Proti čemu

Útočníkovi musí být za každou cenu známa adresa jaderné funkce nebo globální proměnné. Hardcode = funguje jen v jedné verzi.

Kde končí: z Ring 0 (náš driver) často stačí jedna **NtQuerySystemInformation**, nebo prostě scan fyzické paměti po podpisu kernelu.

V PoC použijeme skenování fyzické paměti po MZ hlavičce.

MITIGACE · SMEP / SMAP

Hranice mezi user a kernel

SMEP

Supervisor Mode Execution Prevention

Kernel nesmí vykonávat stránky s U/S bitem user. Nelze skočit do shellcode nahraného v user heapu.

SMAP

Supervisor Mode Access Prevention

Kernel nesmí ani **číst/zapisovat** user stránky bez explicitního **stac/clac**. Vylučuje „kernel přečetl můj shellcode".

Kde končí: útok nespoléhá na kernel-skok do user kódu. Když získám primitiv pro přepis kernel paměti, SMEP/SMAP se mě netýká.

Náš útok nikdy neskočí z Ring 0 do Ring 3 kódu.

MITIGACE · DSE + PATCHGUARD

Nesmíš nahrát ani přepsat

DSE

Driver Signature Enforcement

Jádro odmítne nahrát nepodepsaný driver. „Jen Microsoft a jeho partneři“ smí do Ring 0.

> **Obchází se:** přinesu si podepsaný, ale **zranitelný** driver = BYOVD.

PatchGuard

Kernel Patch Protection

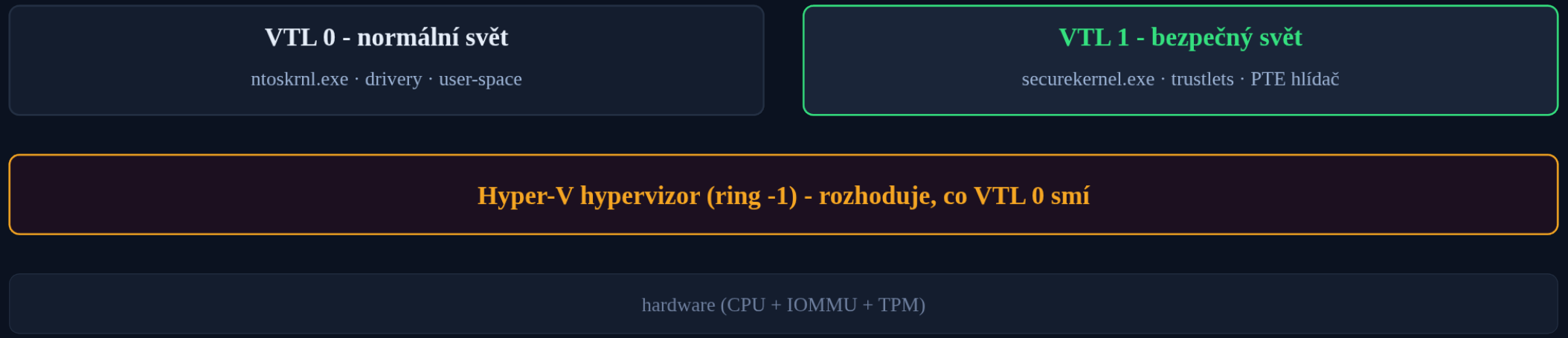
V náhodné časy kontroluje hash kritických struktur: SSDT, IDT, kernel kód. Změna = BSOD.

> **Obchází se:** nic trvalého se nepřepisuje, mění se jen „datová" pole EPROCESS - Token.

BYOVD + token swap = elegantní cesta kolem obou.

MITIGACE · HVCI / VBS

Hypervizor nad jádrem



HVCI (Memory Integrity) znamená: ani kernel sám nemůže udělat stránku **zapisovatelnou a spustitelnou zároveň**. Správu PTE kontroluje hypervizor.

HVCI brání kernel code injection - ne data-only útoku jako token swap.

ČÁST 3 / 6

Windows drivery

Jak user-space mluví s jádrem, co je IOCTL a **kde se dá v tom rozhraní udělat chyba**.

~15 min

DRIVERY · CO TO JE

Ovladač = kód v Ring 0

PE soubor s příponou .sys

- Formát jako .exe, jen cíl je kernel
- Importuje **ntoskrnl.exe** a HAL
- Vstupní bod: **DriverEntry**
- Musí být podepsaný (DSE)

K čemu slouží

- HW ovladače (síťovka, GPU…)
- Souborové systémy (NTFS, ReFS…)
- Antiviry, EDR, DRM
- Služby, co potřebují Ring 0

Pozor: jakmile driver je v jádru, nemá nad sebou žádnou záchrannou síť. Bug v ovladači = bug v jádru.

A to platí i pro ktapi.sys.

DRIVERY · NAHRÁNÍ

Jak se driver dostane do jádra



BYOVD útočník: má admina → krok 1-3 udělá legitimně, krok 4 projde (driver je skutečně podepsaný), v kroku 6 driver sám registruje své zranitelné IOCTL.

Nahrání vyžaduje admin práva - ale PE je typicky z admina na SYSTEM.

DRIVERY · DEVICE + SYMLINK

Jak se k driveru připojit

DeviceObject	Jádrová struktura, kterou si driver vytvoří přes **IoCreateDevice**. Má jméno v NT namespace, např. **\Device\ktapi**.
Symlink	Odkaz z user-space Win32 namespace: **\DosDevices\ktapi** → **\Device\ktapi**. Vytvoří **IoCreateSymbolicLink**.
User-space	**CreateFileA("\\\\.\\ktapi", ...)** → handle, se kterým voláme IOCTL.

Z pohledu user-space je driver obyčejný soubor.

DRIVERY · IRP

IRP: I/O Request Packet

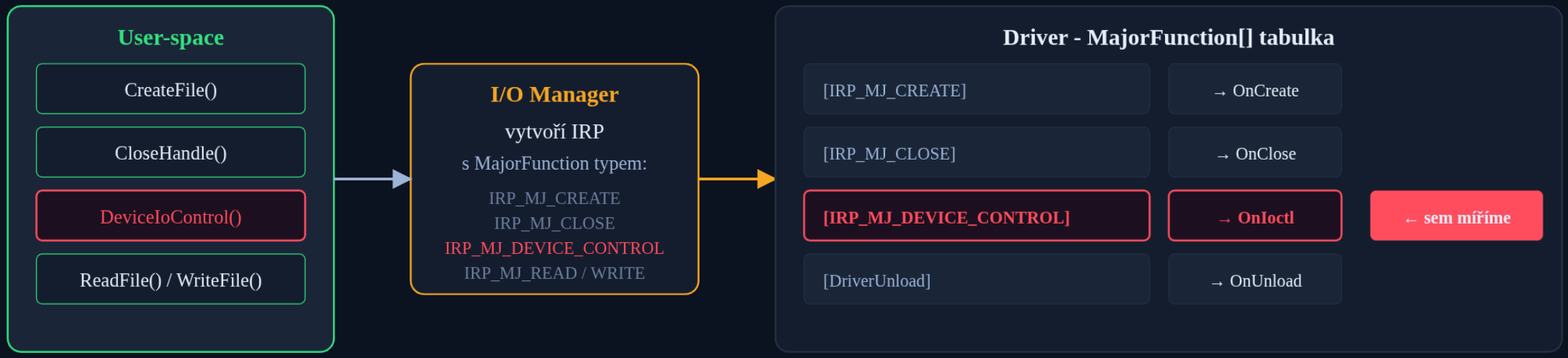


Zranitelnosti ovladačů žijí typicky v OnDeviceControl.

DRIVERY · MAJOR FUNCTIONS

Mapa IRP → callback

Jádro nekontaktuje driver přímo. Všechno jde přes **I/O Manager**, který vytvoří **IRP** a doručí ho podle typu.



Přesně tahle struktura nás čeká v DriverEntry ktapi.sys.

DRIVERY · IOCTL

Co je IOCTL kód

32bitové číslo, které říká: „jaký druh zařízení, jaká funkce, jak přenést buffer, kdo smí volat“.

DeviceType	Access	Function	Method
bity 31-16	bity 15-14	bity 13-2	bity 1-0
např. FILE_DEVICE_UNKNOWN	R / W / RW	libovolný význam, nad 0x800 user-defined	BUF / IN / OUT / NEITHER

> V reverzingu vidíme kód jako konstantu typu **0x22E004**. Switch v OnIoctl rozhoduje, kterou interní funkci driver zavolá.

Method = jak se přenese buffer. Zásadní pro bezpečnost.

DRIVERY · IOCTL METHOD

Čtyři způsoby přenosu bufferu

Method	Jak	Riziko
METHOD_BUFFERED	Jádro user data zkopíruje do vlastního bufferu.	Bezpečné; pomalejší pro velké buffery.
METHOD_IN_DIRECT	User buffer zamčen v paměti, předán jako MDL.	OK, pokud driver respektuje R/W flagy.
METHOD_OUT_DIRECT	Totéž, jen pro zápis zpět.	OK, pokud driver respektuje R/W flagy.
METHOD_NEITHER	Driver dostane raw user pointer.	Žádná validace. Odtud chyby pramení.

U ktapi.sys: kódy končí na **0x00**, takže metoda je **METHOD_BUFFERED**. Bug není v metodě - driver bezpečně dostane zkopírovanou strukturu a stejně v ní předané adrese slepě věří.

METHOD_BUFFERED není samo o sobě bezpečné - záleží na obsahu.

ČÁST 4 / 6

Útoky na drivery

Plocha útoku, BYOVD, LOLDrivers a **kdo to v reálu používá**.

~15 min

ÚTOKY · PLOCHA

Co driver útočníkovi nabízí

arbitrary RW

Zápis/čtení na libovolnou adresu v jádru. Nejsilnější primitiv.

phys mem RW

Totéž, ale cez fyzickou adresu. **Náš případ.**

MSR access

Čtení/zápis Model Specific Registers (např. SYSCALL target).

arbitrary call

Zavolat libovolnou kernel funkci. Vzácnější.

handle duplication

Získat handle na libovolný proces s PROCESS_ALL_ACCESS.

registry / FS

Driver obchází ACL. Vhodné pro ukrytí stop.

Fyzická paměť R/W je mezi nimi nejmocnější - vidí všechno.

ÚTOKY · PŮVOD

BYOVD vs. už nainstalovaný

BYOVD

Útočník si zranitelný driver **přinese sám**.

+ funguje všude, kde platí podpis
+ obchází DSE
- potřebuje admin pro nahrání
- driver může být v blocklistu

Pre-installed

Driver je už na stroji - antivir, HW tool, GPU.

+ nic nenahrává = minimum stop
+ často bez admina
- vázané na konkrétní vendor
- aktualizace díru opraví

Náš scénář: klasická BYOVD - admin si přinese ktapi.sys, nahraje ho, zranitelnost zneužije k PE na SYSTEM. Potřebujeme pak už jen podpis **bez** blocklistu.

Admin → SYSTEM: hranice, kterou Microsoft oficiálně nebrání.

ÚTOKY · CÍLE

Co se tím dá udělat

A

Privilege escalation

User/admin → SYSTEM. Dnes typicky token swap v EPROCESS. **Náš případ.**

B

Vypnutí ochrany

Smáznout callback EDR, odpojit mini-filter, zabít proces s PP ochranou. **Oblíbené u ransomwaru.**

C

Perzistence

Nainstalovat vlastní driver jako „legitimní“, přežít reboot a aktualizaci.

V reálu se tyhle kategorie **kombinují**: ransomware nejprve BYOVD vypne EDR, pak teprve šifruje. Red teamy to taky používají dohromady.

V PoC se zaměřujeme na A - privilege escalation.

ÚTOKY · LOLDRIVERS

Katalog zranitelných driverů

loldrivers.io je komunitní databáze podepsaných, ale zranitelných Windows ovladačů. Pro útočníky je to katalog, pro obránce zdroj pro blocklist.

Pro každý driver nese

- SHA-256 hash binárky
- Jméno vendora a produktu
- Platný podpis (ano/ne)
- Popis zranitelnosti
- Taktiky MITRE ATT&CK
- Odkaz na PoC, pokud existuje

`ktapi.sys`

Vendor: Korton
Podpis: platný, cross-signed
Primitiv: phys mem R/W
Veřejná analýza: žádná - LOLDrivers nese jen hash

Proto jsem si ho musel vyreverzovat sám.

loldrivers.io · github.com/magicsword-io/LOLDrivers

ÚTOKY · REÁLNÉ NASAZENÍ

Není to akademický problém

V článku odkazuji na konkrétní dokumentovaný BYOVD útok. Playbook: „vypnout EDR, pak šifrovat“.

The Gentleman	Ransomware kampaň popsaná **Expel blogem**. Útočníci nasadili ktapi.sys přesně k deaktivaci EDR před šifrováním - stejný primitiv, co ukazujeme my.

ŠIRŠÍ KONTEXT (MIMO ČLÁNEK)

iqvw64e.sys	Intel driver, který byl v době psaní článku dlouho mimo Microsoft blocklist - jeden z typických BYOVD cílů.
Ransomware obecně	BYOVD je standardní krok v ransomware playbooku poslední roky - CISA, Mandiant i jiné reporty to opakovaně dokumentují.

ČÁST 5 / 6 · HLAVNÍ

Případ ktapi.sys

Reverz v Ghidře, page tables, EPROCESS, krádež SYSTEM tokenu **a živé demo**.

~40 min · největší sekce

KTAPI.SYS · PROFIL

Co ktapi.sys je

vendor

Korton

podpis

platný

test prostředí

Win10 22H2 · 19045

zdroj

LOLDrivers

Co nabízí útočníkovi: IOCTL, které namapuje libovolnou fyzickou stránku do user-space na čtení i zápis. Žádná kontrola rozsahu adres.

Žádná veřejná analýza - musel jsem začít od nuly.

KTAPI.SYS · NÁSTROJE

Čím jsem analyzoval

Ghidra

Statická analýza

Open-source disassembler + decompiler od NSA. Nahrát .sys, auto-analýza, ručně projít DriverEntry, IOCTL dispatch a mapovací rutinu.

Žádná dynamická analýza - celý reverzing čistě staticky v Ghidře.

C

PoC exploit

User-space program v C. Win32 API: CreateFileA pro handle na driver, DeviceIoControl pro IOCTL, PSAPI pro kernel adresy.

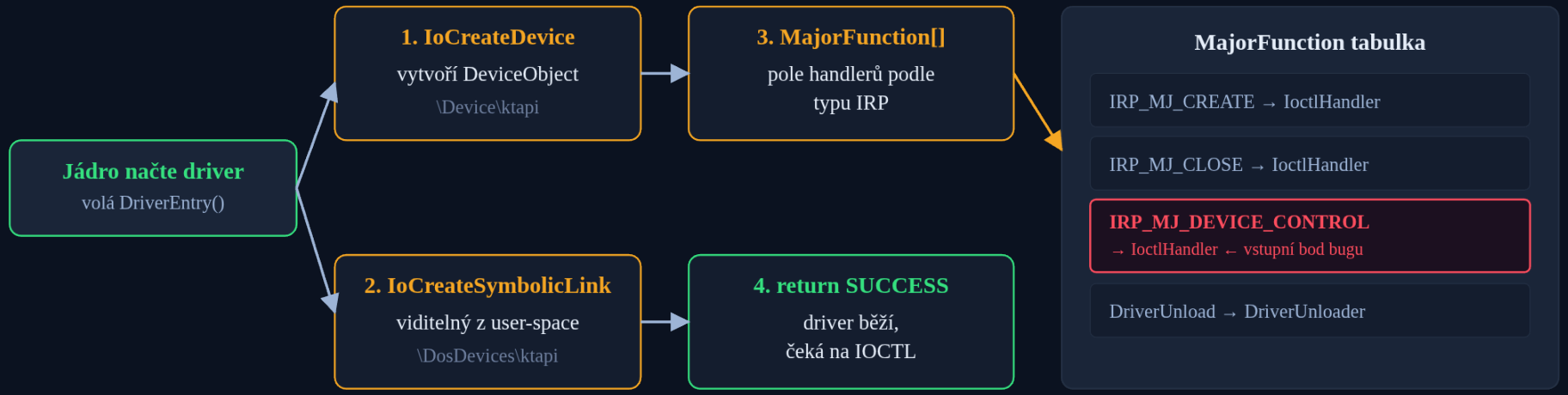
Žádný WDK - útočník nepotřebuje nic navíc, stačí běžný kompilátor.

Ghidra na reverzing, C na exploit. Víc nebylo třeba.

KTAPI.SYS · DRIVERENTRY

Co driver v DriverEntry udělá

Čtyři kroky, přes které se driver představí jádru a **otevře dveře** z user-space.



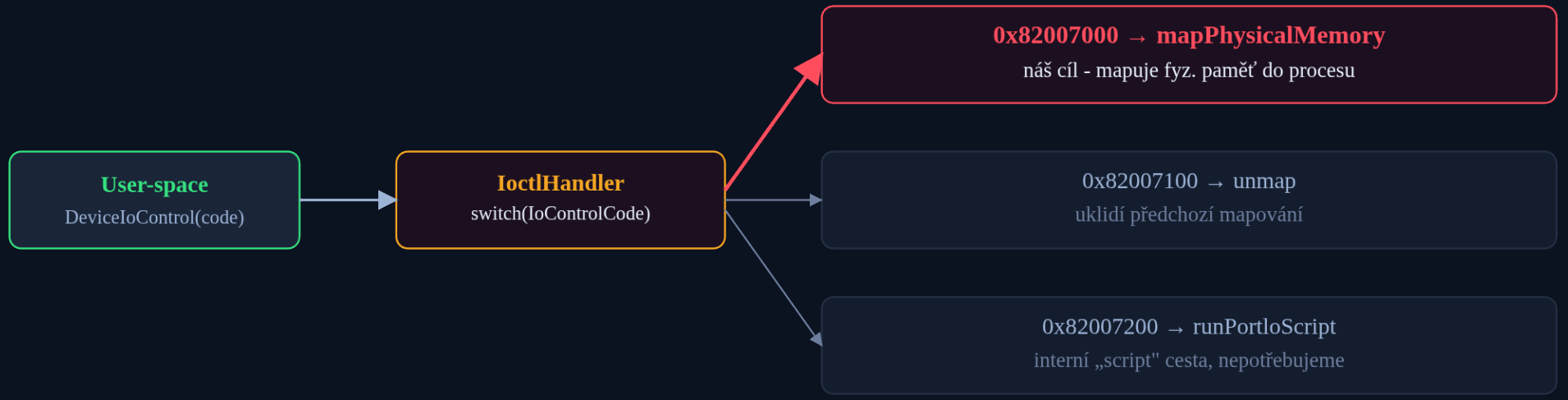
Slabina už tady: device vytvořené bez **FILE_DEVICE_SECURE_OPEN** a bez ACL. Jakmile existuje symlink, otevřít ho může **i neprivilegovaný proces** - driver tiše předpokládá, že útočník má admin.

Nás zajímá jedno pole v tabulce: IRP_MJ_DEVICE_CONTROL.

KTAPI.SYS · DISPATCH

IoctlHandler: tři možné cesty

Dispatch funkce se podívá na **IoControlCode** a **rozhodne, co udělat**. Ktapi má jen tři rutiny.



Žádná access control. Dispatch nerozlišuje, kdo IOCTL posílá - nehledá SID, token ani privilegium. Jediné co rozhoduje je **číslo kódu**.

Náš cíl: kód 0x82007000.

KTAPI.SYS · MAPOVACÍ RUTINA

mapPhysicalMemory: srdce zranitelnosti

Čtyři kernel volání vedle sebe. Driver bere **physAddr** i **length** přímo od útočníka a **bez validace** je předá dál.



Zranitelnost: žádná z těch funkcí sama o sobě nic špatně nedělá. Chyba je, že driver **vůbec nekontroluje**, jestli adresa patří volajícímu, nebo jestli rozsah dává smysl.

`\Device\PhysicalMemory = celá RAM, dostupná z Ring 0.`

KTAPI.SYS · PRAKTICKÉ MEZE

Co driver nehlídá a co se stejně stane

Driver sám

Kontroluje jen **minimum**: vstup **0x18** B, výstup **8** B. Žádný strop na **physAddr** ani **length**.

Praktický strop

HalTranslateBusAddress v mojí lab (VirtualBox) odmítá **> 2 MB** na jedno volání.

Řešení: útočník si napíše vlastní **ReadPhys** / **WritePhys** helpers nad IOCTL - rozsekne čtení na menší bloky.

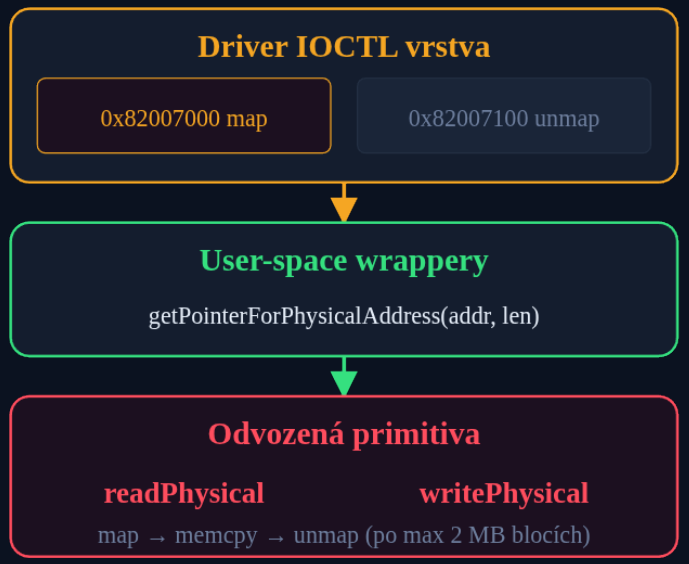
```
pseudokód:
for every chunk ≤ 1 MB of target:
    IOCTL(0x82007000, phys, len) → userVa
    memcpy(dst, userVa + offset, bytes)
    IOCTL(0x82007100, userVa)
```

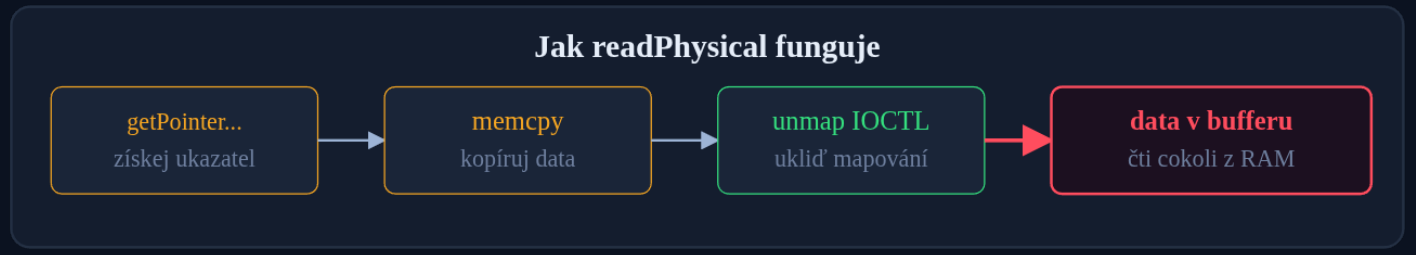
Hranice není v driveru - je ve volaném HAL API.

KTAPI.SYS · PRIMITIVA

Co máme v ruce

Z jednoho IOCTL si postavíme **dvě neomezené primitiva**: čtení a zápis libovolné fyzické adresy.





S těmito dvěma funkcemi + CR3:

Přečíst libovolnou kernel strukturu
Přepsat libovolné místo v paměti jádra
Projít page tables, EPROCESS, tokeny

R/W

celá fyzická paměť

S těmito dvěma funkcemi + CR3 už jsme pány systému.

PŘÍPAD KTAPI.SYS · ČÁST 2

Stavíme PoC

Teorii máme za sebou - teď **kód**.
Od ReadPhys ke SYSTEM tokenu krok za krokem.

pokračujeme ve stejné sekci · cca 20 min + demo

POC · PLÁN

Šest kroků ke SYSTEM

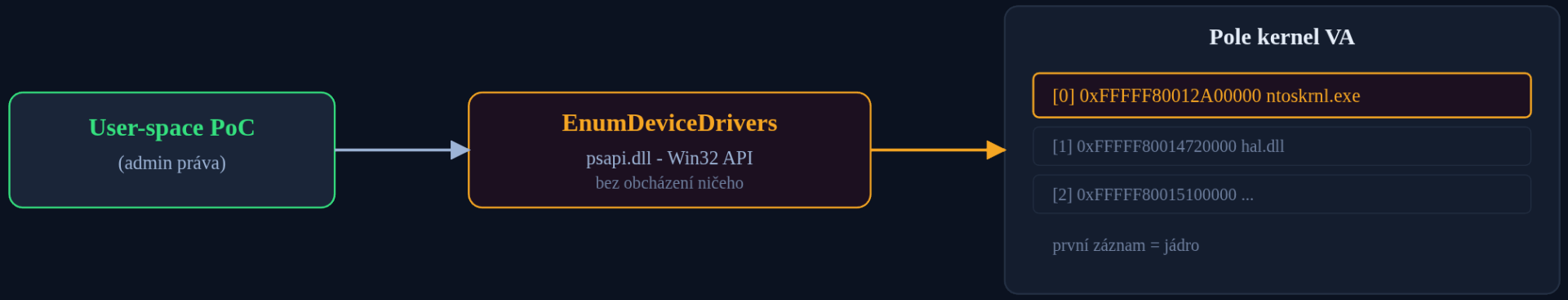
1. **01** Otevřít handle na **\\.\ktapi**
1. **02** Získat **ntoskrnl base** přes **EnumDeviceDrivers**
1. **03** Získat **CR3** ze signatury **HalpLMStub** v Low Stubu
1. **04** Projít page tables = překládat virt -> phys
1. **05** Najít **EPROCESS** SYSTEMu a svůj proces
1. **06** **Překopírovat Token** SYSTEMu do svého procesu

Celé to zabere asi 150 řádků C.

POC · KROK 2

Získat base adresu ntoskrnl

KASLR nás netrápí - když má PoC admin práva, Windows nám adresu driverů **dá sám**, legitimním API.



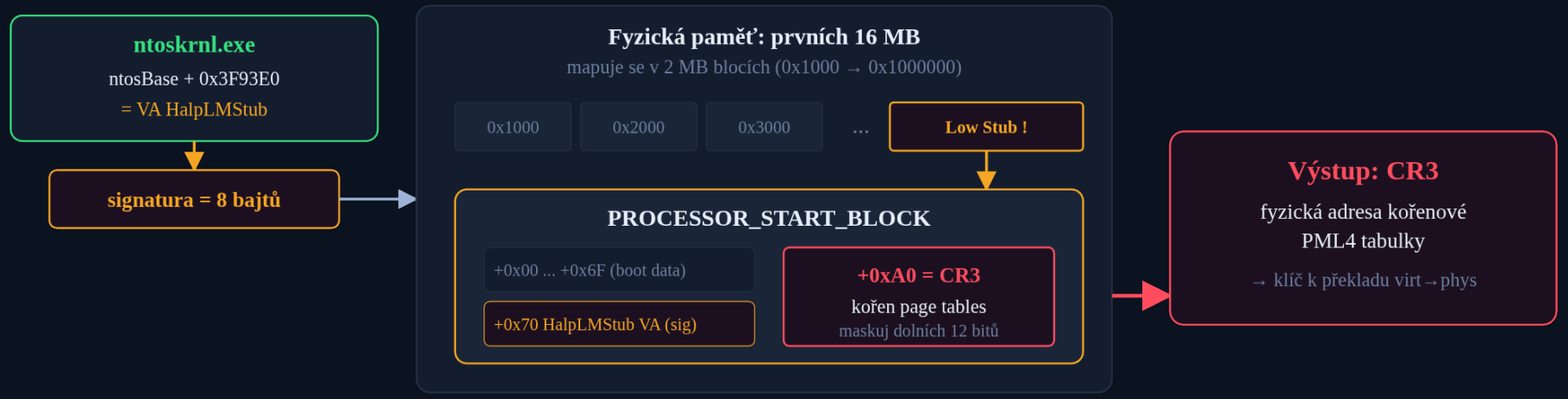
Princip: admin účet má dostatečná práva, aby Windows ochotně vrátil adresy všech kernel modulů. **Žádné hledání v paměti, žádný bypass.**

Trik není v exploitu - trik je použít legitimní cestu.

POC · KROK 3

Najít CR3 přes Low Stub

Při bootu systém odkládá **PROCESSOR_START_BLOCK** do prvních **16 MB** fyzické paměti. Signatura: kernel VA symbolu **HalpLMStub**.



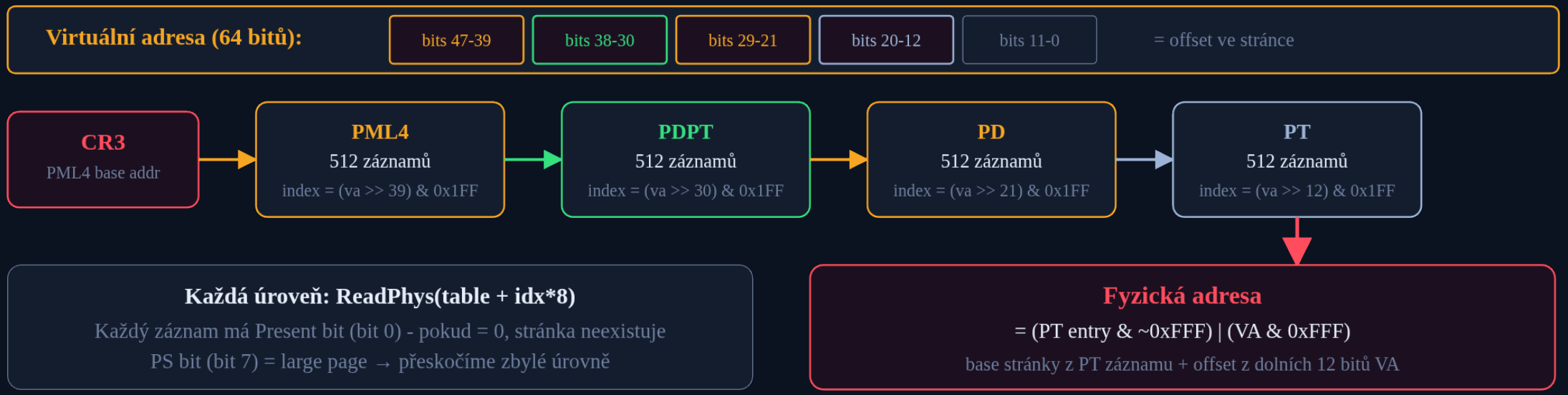
Pozor: offset **0x3F93E0** i **+0xA0** jsou vázané na konkrétní build. Stejnou techniku používá Volatility i rekall pro forenzní analýzu RAM.

Jedna z klasických cest - používá ji i Volatility / rekall.

POC · KROK 4

Ruční průchod page tables

To, co jindy dělá **MMU hardwarově**, teď děláme softwarově: čtyři úrovně, čtyři čtení z fyzické paměti.



To, co jindy dělá MMU hardwarově, teď dělám softwarově.

POC · KROK 5A

EPROCESS: struktura procesu v jádře

Každý proces má v kernelu **_EPROCESS** strukturu. Jsou spojené dvojsměrným seznamem přes **ActiveProcessLinks**.

Offset	Field	Co to je
+0x440	UniqueProcessId	PID (0x4 = SYSTEM, můj PID = jiné)
+0x448	ActiveProcessLinks	LIST_ENTRY - další/předchozí proces
+0x4B8	Token	Ukazatel na access token (SID, privilegia)
+0x5A8	ImageFileName	prvních 15 znaků EXE jména

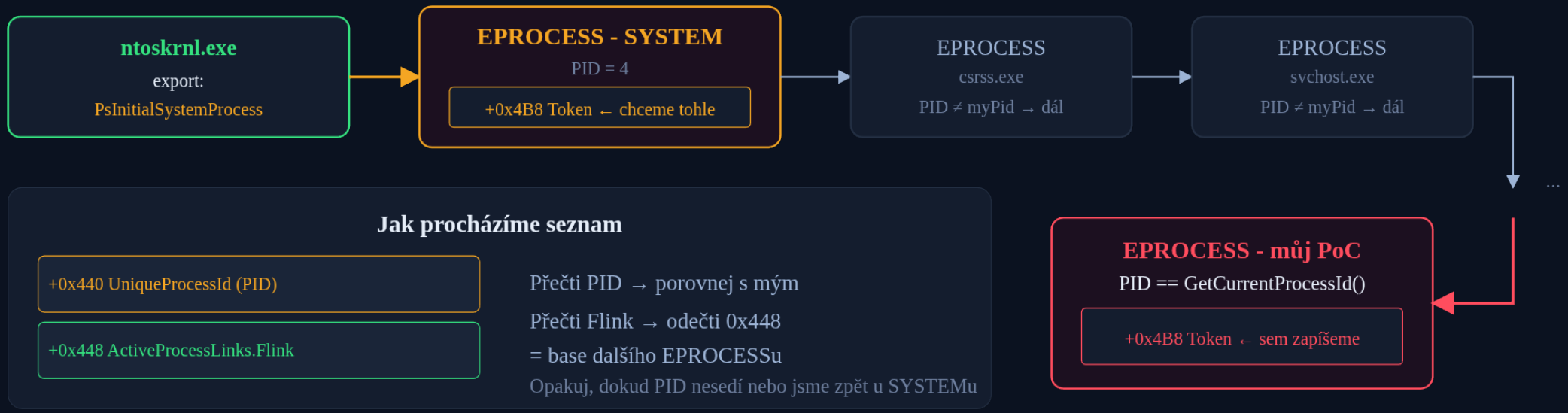
Offsety výše platí pro Win10 22H2 (build 19045). Na jiných verzích jsou odlišné - produkční exploit by je zjistil dynamicky z PDB nebo heuristicky.

Token je ukazatel - to je, co budeme kopírovat.

POC · KROK 5B

Najít System a vlastní proces

Kernel exportuje **PsInitialSystemProcess** - ukazatel na EPROCESS SYSTEMu. Odtud procházíme **ActiveProcessLinks** kruhový seznam.

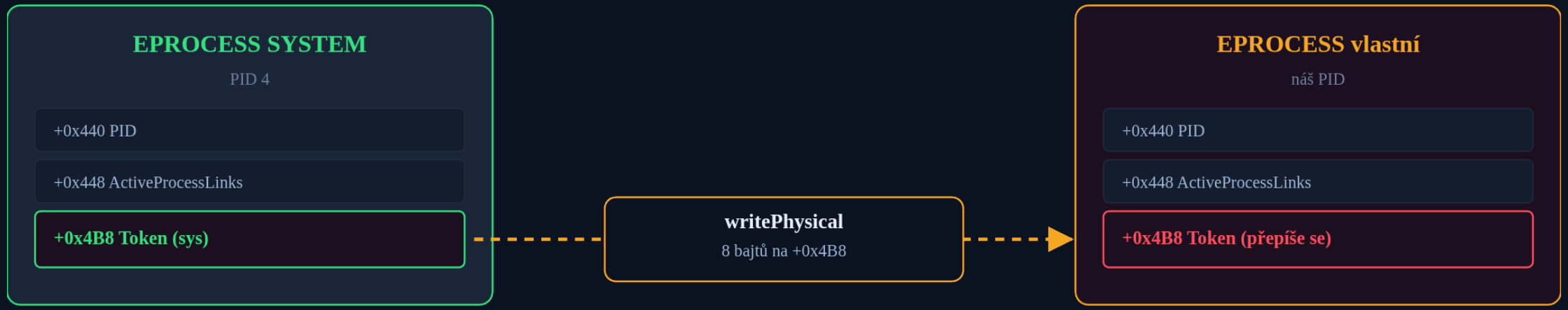


Teď máme virt. adresy obou EPROCESSů.

POC · KROK 6 · FINÁLE

Token field: přesměrujeme ukazatel

V obou EPROCESS leží na stejném offsetu **Token** - ukazatel na access-token objekt. Jedna zapsaná 8bajtová hodnota změní identitu procesu.



Dolní 4 bity: flagy **EX_FAST_REF** (reference count). PoC je **maskuje** (**0xFFFFFFFFFFFFFFF0**) - zapíše čistý ukazatel bez ref-countu.

Osm bajtů = celý výsledek téhle přednášky.

POC · ŽIVÉ DEMO

A teď naživo

1. Windows 10/11 VM, HVCI vypnuté, admin shell

2. **sc create** + **sc start ktapi**

3. **whoami** → BUILTIN\Administrator

4. spustím **poc.exe** → nová konzole → **whoami** → **NT AUTHORITY\SYSTEM**

V případě pádu demo-boha mám připravené screenshoty.

ČÁST 6 / 6

Obrana

HVCI, WDAC a LOLDrivers blocklist. **Co pomůže a co ne.**

~10 min

OBRANA · HVCI

Memory Integrity (HVCI)

Co to brání

Ani kernel sám neudělá stránku zároveň **writable + executable**. Hypervizor kontroluje úpravy PTE.

Zabíjí celou třídu útoků typu „nainjektuj do jádra svůj shellcode a spusť ho“.

Co to nebrání

Zápis do **datových struktur** jádra. Náš token swap na HVCI projde - měníme datový ukazatel.

HVCI silně zpevňuje stroj, ale sám přesně náš útok nezastaví.

Doporučení: HVCI zapnout. Je součástí Windows 11 defaultně na kompatibilním HW. Ztrácí se trochu výkon, zisk v bezpečnosti je nesouměřitelný.

Zabraňuje zavedení vlastního kernel kódu - ne token swapu.

OBRANA · WDAC + BLOCKLIST

Nenechte driver vůbec nahrát

WDAC

Windows Defender Application Control

Policy, která definuje, co se smí spouštět / nahrávat. Driver mimo policy = odmítnuto, i když je podepsaný.

MS Blocklist

Microsoft Vulnerable Driver Blocklist

Vestavěný seznam známých zranitelných driverů. Od Windows 11 default; na W10 ručně.

Realita: blocklist se aktualizuje **pomaleji**, než vycházejí PoC. Ktapi.sys v blocklistu dnes už je - ale v době mé analýzy nebyl.

LOLDrivers blocklist (loldrivers.io) je komunitní doplněk - importujete jako WDAC policy a dostanete širší pokrytí než MS default.

Nejsilnější obrana - útok se vůbec nespustí.

OBRANA · DETEKCE

Všimnout si, že se driver nahrál

Sysmon Event ID 6

„Driver loaded“ - jméno, hash, info o podpisu. Centralizovat do SIEMu, alert na neznámý hash v produkci.

Event Log 7045

„Service installed“ - zachytí **sc create** kroku, kde útočník instaluje driver jako službu.

EDR behaviour

Dobré EDR hlídá, kdy user-space proces začne otevírat známá **\Device*** rozhraní „podezřelých“ driverů.

Hash watchlist

LOLDrivers nabízí hashe v JSON / CSV a hotové WDAC policy. Snadno vložit do detekce.

Útok nelze vždy zastavit - ale vždy musíme vidět, že se stal.

OBRANA · MEZE

Kde ani plně aktualizovaný systém nestačí

1. Blocklist je **reaktivní** - nové zranitelné drivery se objevují rychleji než se do něj dostávají. Období „nikdo neví“ je příležitost útočníka.

2. HVCI stále **není defaultně zapnuto** na W10 a na mnoha W11 instalacích s nekompatibilním HW nebo starým ovladačem.

3. Admin → SYSTEM je **Microsoftem oficiálně nehraničené**. Jakmile útočník má admin, exploitem driveru dostane SYSTEM „legitimně“.

4. EDR vendory občas sami přispívají: **jejich vlastní drivery** se čas od času ukážou jako zranitelné.

Obrana v hloubce: HVCI + WDAC + monitorování + hygiena práv.

ZÁVĚR

Shrnutí & Q&A

Co si z toho odnést, kam dál a **prostor na dotazy**.

~5 min

ZÁVĚR · CO SI ODNÉST

Čtyři věci k zapamatování

1. Podpis driveru je slabá záruka. „Kdysi podepsal důvěryhodný vendor“ ≠ „bezpečný dnes“.

2. BYOVD je mainstreamový krok - ne akademická kuriozita. Ransomware ho používá.

3. Čtení/zápis fyzické paměti je nejsilnější primitiv - obchází izolaci procesů i většinu paměťových ochran.

4. Obrana: HVCI + WDAC + monitorování. Žádná jediná vrstva nestačí - stavte to v hloubce.

Kdo si z toho odnese tyhle čtyři, splnil přednášku svůj účel.

ZÁVĚR · ZDROJE

Kam dál

Můj článek

`hron.re/cs/article/ktapi-lpe`

Celé v textu, s Ghidra screenshoty.

PoC repository

`github.com/n00bDebugger-git/ktapi-exploit-PoC`

Zdrojáky, buildovatelné.

LOLDrivers

`loldrivers.io`

Databáze + API, WDAC/Sigma/YARA signatury.

Expel: The Gentlemen

BYOVD case study - konkrétní reálné nasazení, které v článku odkazuji.

Windows Internals

Russinovich, Solomon, Ionescu - 7. edice. Reference pro vše kolem jádra.

MS Driver Blocklist

`aka.ms/VulnerableDriverBlocklist`

Oficiální MS seznam + návod pro W10.

Slajdy po přednášce na hron.re.

DĚKUJI

Dotazy?

Článek

hron.re/cs/article/ktapi-lpe

PoC repository

github.com/n00bDebugger-git/ktapi-exploit-PoC